A HYBRID COMMUNICATIONS NETWORK SIMULATION-INDEPENDENT TOOLKIT

THESIS

David M. Dines, BA
Major, USAF

AFIT/GCS/ENG/08-08

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCS/ENG/08-08

A HYBRID COMMUNICATIONS NETWORK SIMULATION-INDEPENDENT TOOLKIT

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science (Computer Science)

David M. Dines, BA

Major, USAF

March 2008

AFIT/GCS/ENG/08-08

A HYBRID COMMUNICATIONS NETWORK SIMULATION-INDEPENDENT TOOLKIT

David M. Dines, BA
Major, USAF

Approved:

_____/Signed/_____	_6 Mar 08_
Kenneth M. Hopkinson, PhD (Chairman)	Date
_____/Signed/_____	_6 Mar 08_
Jeffrey T. "Todd" McDonald, PhD (Member) Lieutenant Colonel, USAF	Date
_____/Signed/_____	_6 Mar 08_
Stuart H. Kurkowski, PhD (Member) Lieutenant Colonel, USAF	Date

AFIT/GCS/ENG/08-08

Abstract

Net-centric warfare requires information superiority to enable decision superiority, culminating in insurmountable combat power against our enemies on the battlefield. Information superiority must be attained and retained for success in today's joint/coalition battlespace. To accomplish this goal, our combat networks must reliably, expediently and completely deliver over a wide range of mobile and fixed assets. Furthermore, each asset must be given special consideration for the sensitivity, priority and volume of information required by the mission. Evolving a grand design of the enabling network will require a flexible evaluation platform to try and select the right combination of network strategies and protocols in the realms of topology control and routing. This research will result in a toolkit for ns2 that will enable rapid interfacing and evaluation of new networking algorithms and/or protocols. The toolkit will be the springboard for development of an optimal, multi-dimensional and flexible network for linking combat entities in the battlespace.

Acknowledgments

I would like to thank my Advisor, Dr Kenneth Hopkinson, for his generosity in time, talent and spirit.

I owe special thanks to my wife and our three beautiful children for their love and patience during my effort to complete this document and the degree program.

David Dines

Table of Contents

List of Figures

Page

List of Tables

A HYBRID COMMUNICATIONS NETWORK SIMULATION-INDEPENDENT TOOLKIT

I. Introduction

As the military moves towards a more information-rich environment on the battlefield, network protocol developers need a mobile networking toolkit that allows rapid prototyping and experimentation of the components needed to make Net-centric Warfare a reality. Existing AFIT research in dynamic topology control [15] and fault-tolerant routing [9] algorithms build individual components for testing on a static network simulation. However, designing mobile networking platforms that effectively combine these components together is extremely difficult. Protocol developers need a simulation toolkit that breaks the mobile networking problem down into a series of modularized components for use in a well-known simulation platform like ns2. This toolkit will enable rapid prototyping of protocol combinations, thereby allowing developers to evaluate which combinations make the most sense for military communication systems of the future.

The project goal is to produce an extensible toolkit for use in modeling and evaluating the intersection of new distributed hybrid network algorithms in the ns2 simulation environment while remaining extendable to other network simulation applications. The design effort will apply well-known concepts in object-oriented software engineering patterns [7][8][12][16] and modular networking architectures [2] [10][13] [17] along with knowledge of the ns2 platform architecture [6][5] to implement a specialized network toolkit for use in ns2. Existing routing and topology algorithms

will be carefully studied and redesigned to meet the programming interface of the toolkit. Future protocol developers will use the API, allowing immediate hot-swappable integration of algorithmic components within the simulation.

I organize this document as follows. Chapter 2 provides a background of software engineering principles and existing source code. Chapter 3 describes in detail the toolkit design decisions. Chapter 4 describes the toolkit implementation and structural metrics. Chapter 5 summarizes the research contributions and suggestions for future work.

II. Literature Review

Chapter Overview

The purpose of this chapter is to introduce the reader to several relevant software development projects that utilized the toolkit concept to implement networking simulations. Previous research is examined to expose design considerations and to demonstrate the usefulness of developing toolkits for network simulations. Next, an exploration of the definition of a software toolkit reveals a general direction for the design process. Finally, a brief review of software design patterns is presented as they are the basis of maintainable and extensible programs and present excellent opportunities for toolkit implementation.

Relevant Research

Software research and development efforts focus on building and using toolkits in distributed communications systems. From these sources, one can gain an understanding of this development domain and garner ideas for implementing a successful toolkit.

synTraff

Balakrishnan and Williamson [1] present a suite of toolkits for analysis, modeling and generation of long-range dependent network traffic called synTraff. Each toolkit in the suite provides a common Tcl/TK interface to a set of related C/C++ programs for traffic generation and analysis. This GUI approach allows a user to select from a wide range of network settings to generate a specific model of network traffic. Figure 1 depicts the GUI set for synTraff.

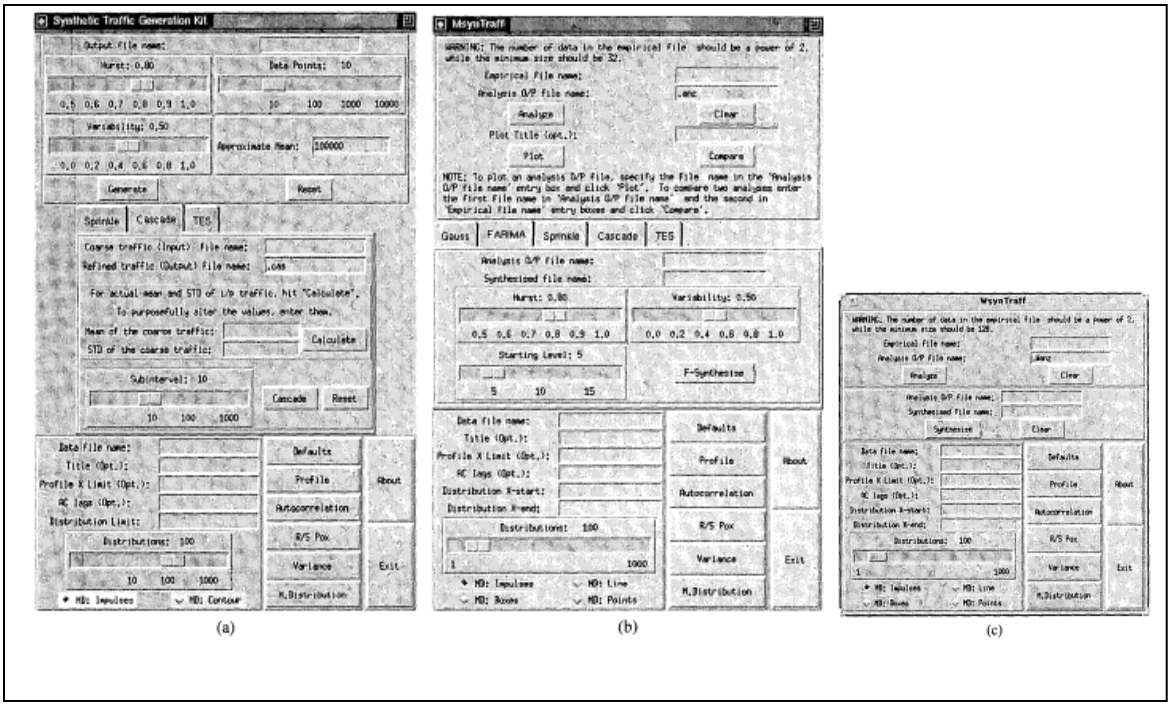

Figure 1. GUI for SynTraff Suite of Toolkits [1]

A key advantage for SynTraff is its approachability by users who are not well versed in software implementation. A protocol developer can use the toolkit to implement the affects of new protocol on network traffic and present that protocol as a network setting for the GUI. A researcher may then apply the protocol easily by turning it on or off, or swapping it out for another related protocol. Together with other system parameters, the researcher can compare network traffic metrics and draw conclusions without knowledge of the detailed system implementation.

SyncML Development Toolkit

Tong [17] produced a toolkit for the development of the SyncML (a universal data synchronization format based on XML) device management protocol. The purpose

of a device manager is to register, configure and implement independent services on a particular device, such as a cell phones and personal data assistants. A service should come packaged with a management agent to allow the service to operate on the device.

The toolkit automates the design and implementation of a device specific SyncML agent. The target usage is to input the device management tree specification and to output a generic agent skeleton requiring minimal modification for a specific device customization. Thus, a device-specific agent can be obtained by starting with an empty agent skeleton and extending it with a complete device management tree specification.

The SyncML Development Toolkit is essentially a framework builder that issues a base type of device management agent that may be customized through extension to accommodate a particular device. Figure 2 depicts the framework and the extension points for the device manager developer.

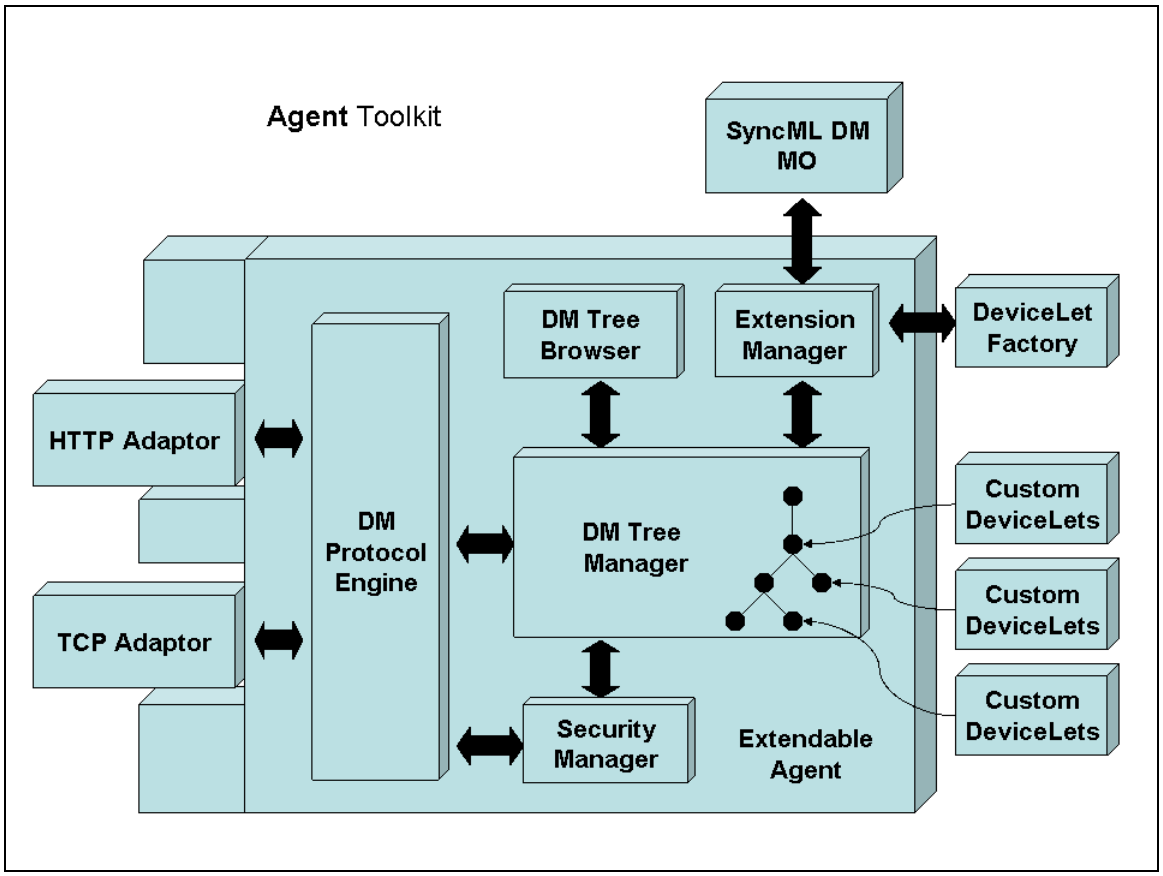

Figure 2. Functional Architecture of the SyncML agent toolkit [17]

WiDS implements Distributed Systems (WiDS)

Shiding Lin, Aiman Pan and Zheng Zhang[13] of Microsoft Research Asia, along with partners from universities at Beijing and Tsinghua, devised a toolkit to optimize the process of developing and testing distributed systems algorithms. The WiDS toolkit adopts an object-oriented and event-driven programming model and provides a small and straightforward set of APIs to support message exchanges and timers. The APIs isolate a WiDS-programmed protocol from any particular runtime that users want to employ. A WiDS object represents a protocol instance or a service. WiDS objects exchange asynchronous messages to each other. Each message is dispatched to the target object's

corresponding handler through the use of an event wheel. In this way, the WiDS architecture allows different runtimes to be shaped by integrating some of the four modules: topology model, networking, system timer and event wheel. **Figure 3** presents a model of this architecture.

The WiDS design offers modularity in a powerful way. The four modules can be independently extended and assembled to address the requirements of a new protocol. If extended polymorphically, the architecture can maintain protocol compatibility. For example, a single Topology model could work with a new protocol while remaining compatible with any other protocol previously developed.

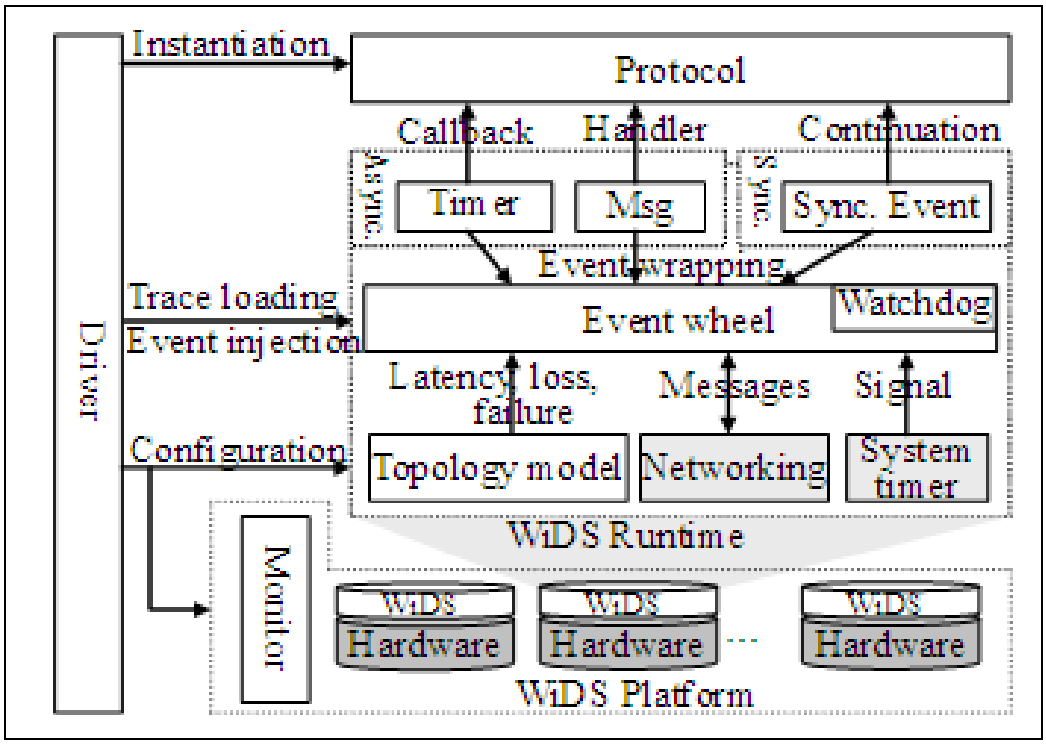

Figure 3. The WiDS architecture [13]

WiDS presents a closed simulation environment where different protocols may be evaluated [13]. However, this implementation does not allow a researcher to investigate

protocol performance across any other established simulation platforms such as ns2 or OPNET.

MarNET

Battenfield [2] presents a modular routing architecture in a single concise interface which facilitates the construction of routing modules that can be used in a particular mobile ad hoc network emulation system MarNET and on any computer system running Linux. The constructed routing modules may be run within the emulator or ported to operational hardware without changing program code or recompiling.

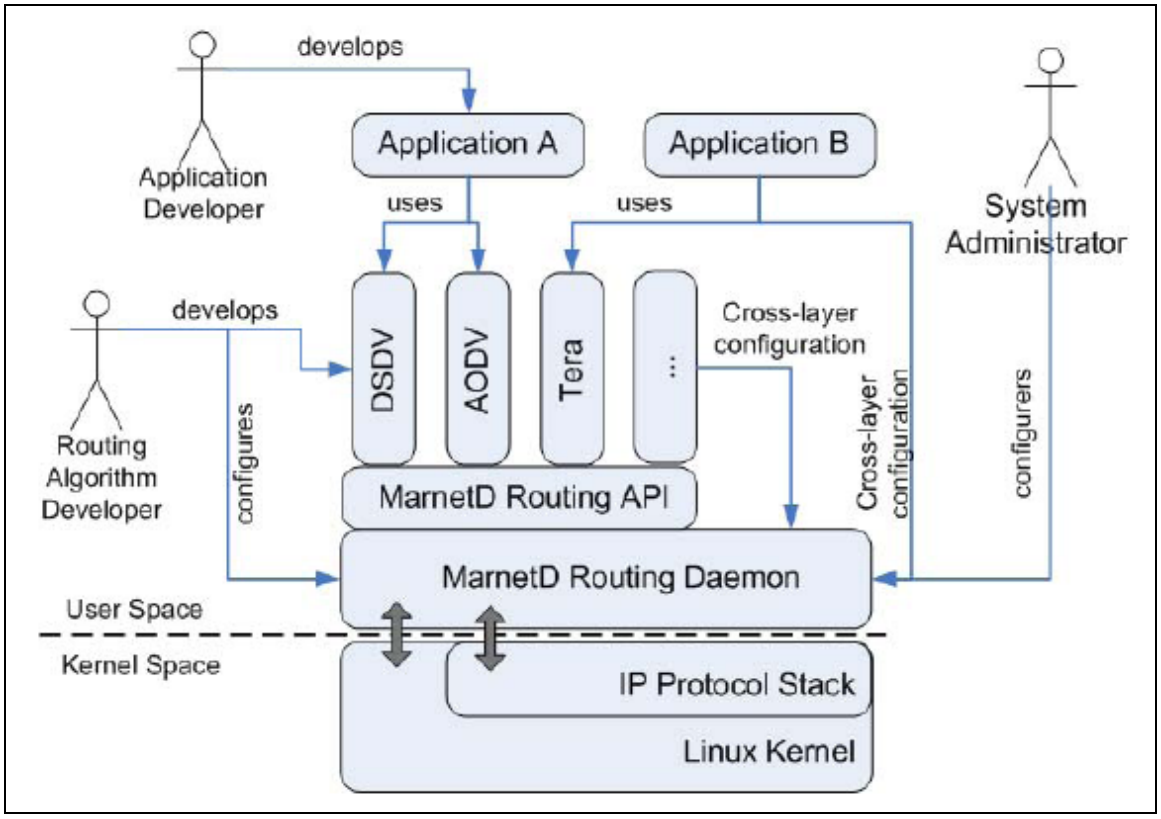

Figure 4. The MarNET Toolkit daemon [2]. MarNET provides the application developer with ready to deploy routing modules. The algorithm developer implements the MarNET routing API along with configuring the routing daemon.

MarNET offers a hierarchical and modular approach for developing and implementing dynamic routing applications as depicted in Figure 4. Further, MarNET offers a portable framework for implementation and evaluation on multiple hardware platforms. However, MarNET does not offer an API explicitly for topology control and can not be used for rapid prototyping of new routing and topology control algorithms. A routing algorithm developer must configure the routing daemon behind the Routing API to ensure operability. As with other simulation toolkits, MarNET employs its own emulator so testing and verification rely on one simulation environment.

Electric Power and Communication Synchronizing Simulator (EPOCHS)

Hopkinson, Wang, et al. [10] present a toolkit called EPOCHS that integrates research and COTS software to provide users a way to evaluate new communications protocols as they relate to electric power control scenarios. EPOCHS smartly links varying existing simulation engines using only their built-in APIs. Further, the system is agent-based which hides the complexity of the combined system, making it easy for users to design new power scenarios involving communication. The system is based on a publish/subscribe mechanism, where any simulation entity subscribing to another will receive all of its updated information, whether or not it is needed.

In EPOCHS, users interact with the system through the use of agents. In this context, agents are autonomous, interactive computer programs which may exhibit mobility, intelligence, adaptability, and communication. Agents conform to an API that hides the differences between the various simulators and essentially makes it easier for users to implement EPOCHS. The team developed AgentHQ to act as a proxy between agents to facilitate agent interaction.

A runtime infrastructure (RTI) routes all messages between simulation components and ensures the simulation time is synchronized across all components. Figure 5 presents the EPOCHS component relationships, note that NS2, PSLF and PSCAD/EMTD are the existing simulators that EPOCHS combines.

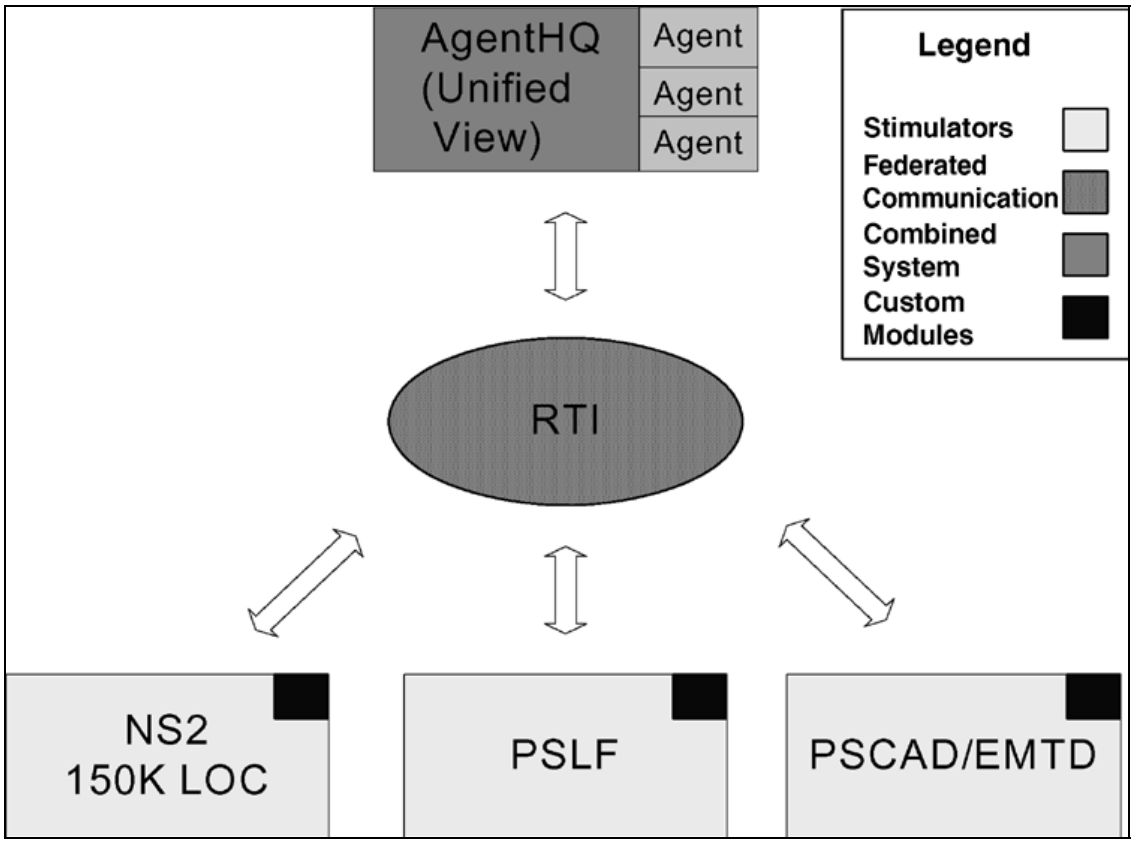

Figure 5. EPOCHS Component Relationships [10]

The RTI presents a single view of a generic simulation engine. This is a powerful concept that allows the development of networking protocols without knowledge of a specific simulation environment. Furthermore, any protocol developed to the RTI may be implemented in any simulation environment that has a custom module developed for it. In EPOCHS, an expert in one simulation environment can develop a custom module for the RTI independent of any specific protocol written to the RTI.

While EPOCHS was developed for electric power control applications, the simulator independent nature of the implementation as enabled by the RTI proves an interesting model for a network simulation-independent toolkit.

Software Engineering Principles

Development of a network simulation toolkit relies heavily on a clear definition of what a software toolkit should be and a solid understanding of flexible and extensible implementation techniques collectively known as design patterns.

Software Toolkits

> **tool** (‘tül, noun): 1a. a handheld device that aids in accomplishing a task. 2a. an instrument or apparatus used in performing an operation or necessary in the practice of a vocation or profession. 2b. an element of a computer program that activates and controls a particular function. [4]

Tools allow people to quickly and easily perform a task that would otherwise be time consuming, laborious or impossible to perform. Tools in software development are useful to abstract specific functions, allowing the user to focus on the interface and results of a software component rather than consume time and effort on understanding its design details. The user of a tool needs only the knowledge of when or how to use the tool. A particularly effective tool accomplishes a specific task, is easy to implement and easy to validate. Furthermore, a software tool may need to accept adjustments as with a torque wrench, to meet unique and specific requirements. For example, a software tool

for implementing a graphical window may have many parameters to aid in adjustment of shape, size, color and contents.

A toolkit is a collection of tools that work in concert to perform a range of tasks within a specific field of endeavor. For example, a dental hygienist's toolkit would contain a variety of medical instruments needed to maintain healthy teeth and gums. However, if the concept of a toolkit is scoped too large, such as "medical toolkit", the toolkit could never be assembled in a useful way since the whole medical field is as wide as it is deep.

Likewise, a toolkit in software is a set of general, yet related functions that can be called in some combination to implement a specific software function. For example, a GUI toolkit provides a set of general graphical elements for an application designer to draw from. The graphical elements, such as buttons or scroll bars, are available as objects that may be adjusted and extended to meet the specific needs of the designer. A toolkit enables sound implementation by leading the developer toward code reuse [12].

There is no standard way to implement a toolkit in software. A toolkit could be a single class that provides instances of a related set of objects. Alternatively, Booch [3] states that a software toolkit is a general purpose library of predefined classes that may be instantiated or extended in user-defined combinations to serve a specific requirement. A toolkit is useful in a particular application domain; however a good toolkit will not impose a particular design on the user.

One example from Booch's definition is the Java collections library [16]. This library offers a base class called Collection that defines the abstraction of a collection of

objects. Other classes provide a variety of specific collection types that may be implemented in general ways by a programmer.

Further, toolkits can be geared toward use within development frameworks. A framework is an architectural pattern that provides an extensible template for applications within a domain [3]. For example, a framework can be geared toward providing an interactive animation for first-person video games. Many unique video games could implement the framework since each game would customize the graphics, interactions and game rules. A framework could be provided as an architectural foundation designed to use a toolkit for customization. To do this, the framework would encapsulate the skeleton of the overall application design and the toolkit would offer a variety of ways to customize, control and implement the framework. This framework-centered toolkit is obviously limited in its applicability to the framework itself, but can still empower its user within the application domain.

A toolkit for network protocol simulation development will provide the components of a network object model offering a variety of ways to define the network and its protocols. A simulation framework must be provided to ensure consistent transition of the simulation through user-defined and framework-provided events.

Design Patterns

Software design patterns have emerged as crucial tools for developing maintainable software. Design patterns reinforce object-oriented design principles, offer a standard for communicating architectural solutions and provide implementation strategies for common software design problems [16].

HCNeT design relies on several design patterns as described by Gamma, Helm, Johnson and Vlissides [8], collectively known as the “Gang of Four”. The following paragraphs provide definitions and summaries of the design patterns that will be referred to by name throughout this paper.

Decorator

The intent of the decorator pattern is to add additional state and behaviors to an object at runtime. This allows an application to handle new responsibilities as they arise. Likewise, with the decorator pattern, an object can shed or ignore particular state and behaviors whenever necessary. In this sense, the decorator is also known as a “wrapper” that can be applied and removed from the object depending on which decoration is required during runtime.

The decorator pattern should be useful for a toolkit that must offer a flexible and dynamically changing definition of a particular object’s state.

Visitor

The visitor pattern offers a way to traverse an object structure while performing operations along the way. Each object in the structure must accept the visitor, and then in turn pass its reference back to the visitor. A traversal happens when one upper-level object accepts the visitor, and then forwards the visit request to the next object in the structure. This behavior is repeated until all objects in the structure are visited.

Applied correctly, the visitor pattern decouples the object structure from an object that must operate on the structure in unexpected ways. Furthermore, the visitor pattern increases cohesiveness of the members of the object structure, by allowing other objects

to handle distinct yet unrelated operations. Finally, visitor provides a class design that is smartly built for frequent extension.

Extensibility is a desirable feature for a software toolkit. Furthermore, The visitor pattern could work well with a network structure composed of interlinking nodes and with object decorators. Recall, the decorator pattern represents a layering of wrappers in a hierarchical composite object structure.

Strategy

The intent of the strategy pattern is to make interchangeable the variations within a family of algorithms. That is, if an object has a particular job to do and many different methods to do that job, then encapsulate each method while hiding how the job gets done.

The strategy pattern makes sense for network nodes that are responsible for routing and the way the routing is performed may be switched out dynamically in response to environmental conditions. This also makes sense for a network that must implement various ways of performing topology control.

Bridge

The bridge pattern decouples an abstract structure from its implementation. This decoupling allows independent variation of the implementation.

This pattern should be useful to define how a network model interacts with various different simulation frameworks. Once the interaction between a network structure and a simulator is defined, a developer could use a bridge to implement those interactions for a particular simulator.

Abstract Singleton

The abstract singleton pattern is not a name found in textbooks, but it is a combination of the abstract factory, factory method and singleton found in [8]. This pattern presents an abstract interface to a singleton object whose specific implementation is determined by the environment, but unknown to the requester. To implement this pattern, a scaled down Abstract Factory class has a Factory Method called getInstance(). The method getInstance() will check a system variable to determine which implementation of the singleton to return. This pattern isn't necessarily a factory since the varying instantiations of the abstract interface are created at software load time and registered within the abstract factory.

Abstract singleton is a pattern that could be used to create the bridge object reviewed in this chapter. In this way a protocol application will run with any pre-defined simulation framework without need for modification.

Observer

The intent of the observer pattern is to notify many objects when a particular object has changed. The observer pattern also limits coupling by allowing one or more widely different objects to gain access to another object without containing the target object.

An observer pattern may benefit a toolkit that must remain flexible. For example, if a bridge is used to decouple the toolkit from its platform, an observer could decouple the toolkit from the bridge. Furthermore, an observer could be used to help synchronize

the actions between network topology and routing, without the two algorithms accounting for which specific strategies are being applied.

III. Methodology

Introduction

Before undertaking a significant software design and refactoring project, a developer must review sound software engineering principles that may contribute to the project. Further, the existing code base must be analyzed to identify requirements and opportunities for code refactoring.

Existing Architecture and Design

The primary purpose for HCNeT is to allow user-defined combinations of existing network algorithms within the Ns2 environment. Therefore, this development effort requires an understanding of how to extend Ns2. In addition, previous work in routing and topology control were selected by the customer for refactoring into HCNeT with the expressed purpose of mixing and matching these works.

Ns2 Architecture

The network simulator NS is a discrete event network simulator developed at UC Berkeley that focuses on the simulation of IP networks on the packet level [6]. Ns2 is a framework for demonstrating network protocols; the design of Ns2 drives the implementation of any protocol. In this context, a protocol developer must expend considerable effort to learn the framework before implementing the protocol within it. While Ns2 is extensible, it is not effective as a toolkit in itself.

To use NS for setting up and running a network simulation, a user writes a simulation program in OTCL script language. Such an OTCL script initiates an event scheduler, sets up the network topology and tells traffic sources when to start and stop

transmitting packets through the event scheduler. The prevalence of OTCL within Ns2 reveals that, first and foremost, ns2 is a split framework. All objects in Ns2 have an implementation in both OTCL and C++ as shown in Figure 6. This split framework is meant to offer simplicity of OTCL for frequently changing functionality and the efficiency of C++ for static, time-intensive functions [11].

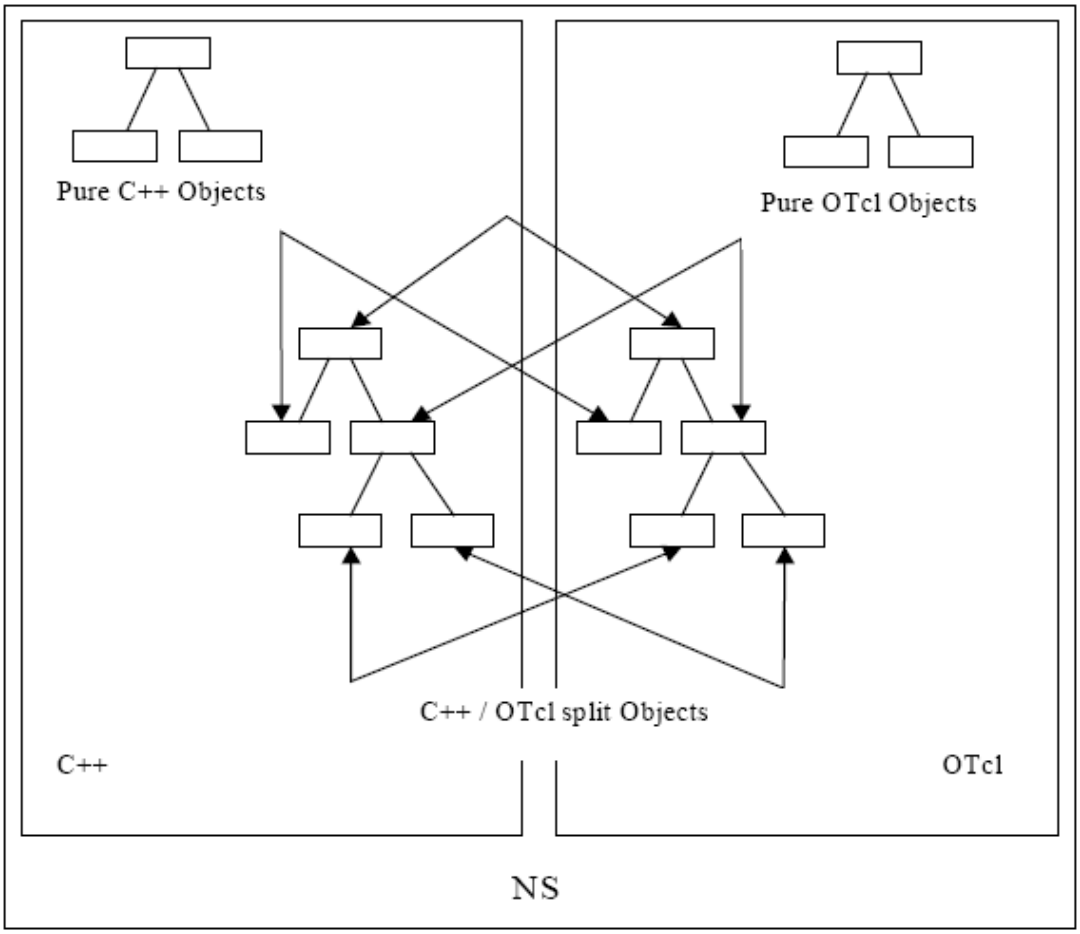

Figure 6. The split architecture of ns2 [6]

The requirements leading to a split framework in Ns2 work against the needs of AFIT protocol developers since they develop protocols to resolve NP-Complete and NP-Hard classes of problems. These protocols are very time-intensive, yet need to be

interchanged and reconfigured often. So, AFIT requirements drive for one environment to handle functions that are both time-intensive and frequently changing.

Another consequence of the split design is the need to create and correctly bind a mirror class for every class a user develops in either side of the framework. As a result, a user intending to implement a protocol must spend time "housekeeping" within the simulation framework before testing the protocol.

Ns2's complexity is further exacerbated by the "openness" of the project. Ns2 has a large community of users with a variety of networking interests and requirements. Some of the existing Ns2 code contains comments alluding to the expeditious workarounds and hacks.

Routing and Topology Control Algorithms

The HCNeT development effort must include a refactoring of existing heuristically driven routing and topology control algorithms. In particular, Garner [9] implemented a purely C++ application for solving the multi-commodity network flow problem allowing user-selected forms of the Pre-flow Push and Edmonds Karp algorithms, both with an alternate version of the Knapsack search algorithm.

In the class design depicted in Figure 7, Garner defined a network model consisting of nodes and edges, but created many interdependencies between the KnapsackOps and EdmundsKarp algorithms and the elements of the network: Commodity, Node and Edge. Additionally, the Network object is contained by each

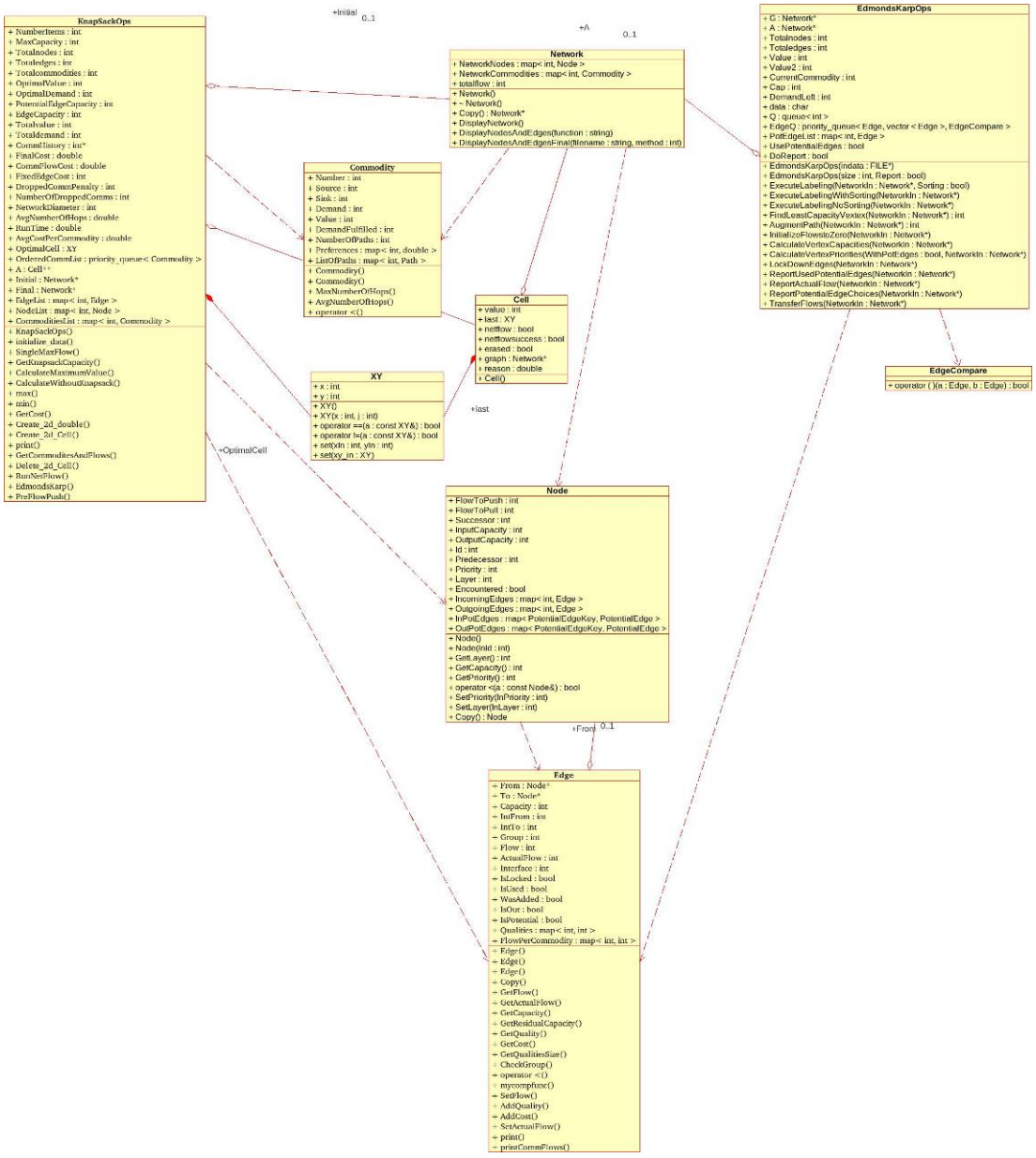

Figure 7. Class model for Garner's routing and topology control algorithm. This model is not easily extendable due to high coupling.

algorithm, whereas this research requires a network model that can switch between multiple algorithms without knowing how the algorithms work.

Since Garner's application does not interface in any way with a simulation engine, it presents an ideal case for refactoring into the HCNeT domain to validate the toolkit design and implementation.

Fault-Tolerant Routing Algorithm

HCNeT must also include the fault-tolerant routing algorithm as designed by Lewellyn [6]. Lewellyn researched topology control and routing in clustered networks. His primary interest was developing an efficient way for network clusters to identify network faults locally, then automatically reroute affected communication links within the cluster.

Llewellyn's implementation is particularly interesting since it utilizes a concept from Hopkinson's agent-based simulation [5] to interface with ns2. Figure 8 illustrates the architecture of Llewellyn's implementation. Note how the RoutingTopologyControl class contains the network model, much like Garner's KnapsackOps and EdmundsKarpOps classes. Again, this design puts emphasis on one particular algorithm which prohibits introduction of new, complementary algorithms.

Note also how the AgentHQAgent class contains the algorithmic class along with a reference to the user-defined model class RTNode. This containment shows how the ns2 agent is highly coupled to the user-defined network model. A developer could leverage the Ns2 implementation of AgentHQAgent to develop a bridge between the network model and the Ns2 simulation environment.

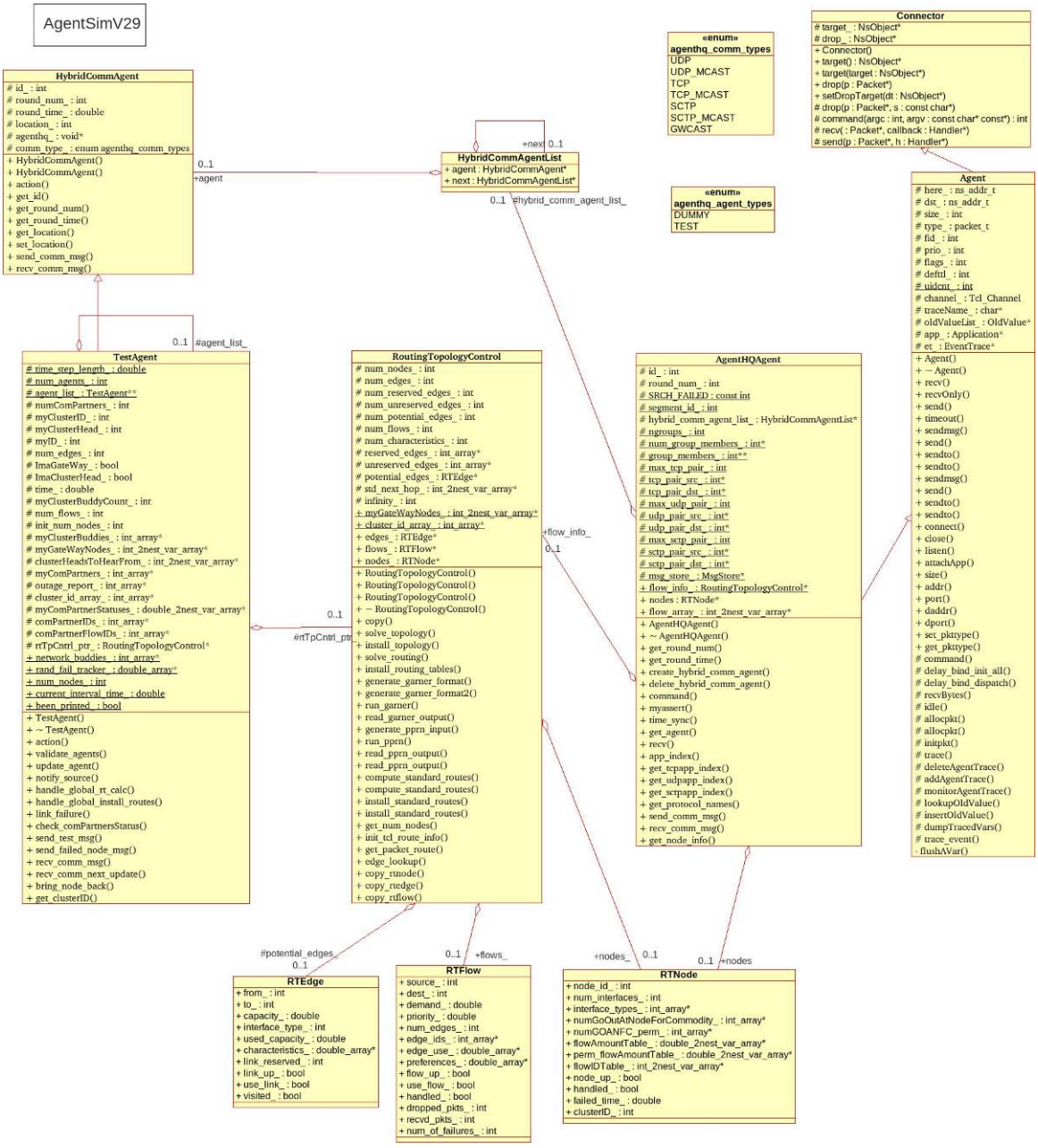

Figure 8. Class model for Llewellyn's fault-tolerant routing algorithm in ns2. AgentHQAgent provides the single interface to the ns2 simulation. This model is not easily extendable due to high coupling and low cohesion.

Hybrid Communications Network Toolkit (HCNET)

Requirements

To begin requirements analysis, the software developer must be familiar with who the customer is. For HCNeT, the customer is the AFIT Hybrid Communications Research Group (HCRG). In this group, students develop new networking protocols based on extensive research. Much of the research is presented at a theoretical level. This leaves a significant implementation gap that must be crossed before evaluating the theory. In many cases protocol developers must devote the majority of their time understanding and extending the simulation framework in which they will evaluate their protocol.

Not only is it difficult for HCRG members to get their protocol off the ground, it is even more difficult to match and compare protocols between members. As revealed in paragraph 2, the software designs are algorithm-centric and not ready for compatible interfacing with other complementary protocols.

AFIT protocol developers call for a toolkit that will allow them to concentrate on the complexity of their algorithms and not on the complexity of implementation within the ns2 simulation framework.

The toolkit needs to provide all the elements of a basic network definition (nodes and edges), plus the ability to extend and add state to the network as required by a particular protocol. Some users may want an abstract, graph-oriented model for high-level protocol implementation. Other users will need node-level extensions to realize their protocol.

The toolkit also requires a simple interface that users will extend to implement the logic of their protocol. In particular, the user requires a routing module and a topology control module that will each operate on the network model.

Figure 9 highlights the use cases that summarize HCNeT requirements. Use case scenarios are documented in Appendix A.

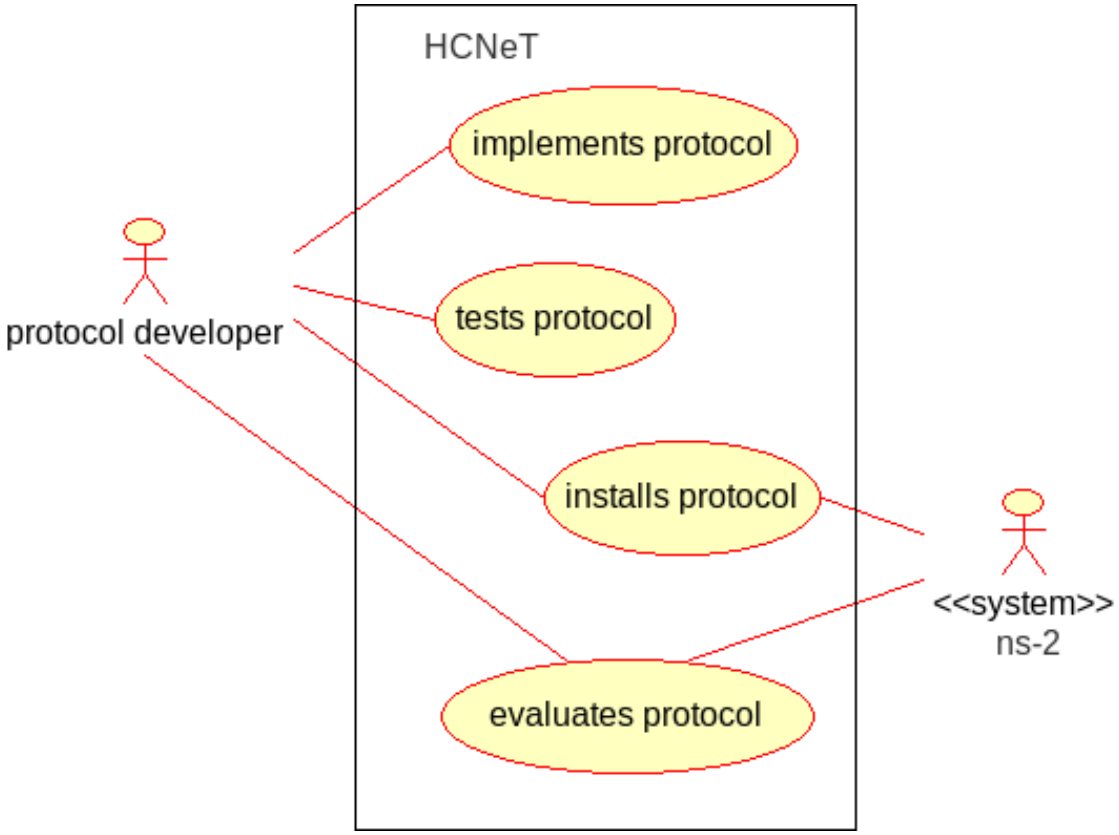

Figure 9. HCNeT Use Case Model. Protocol developers need to implement a protocol that will be hot-swappable and compatible to other complimentary protocols. Implementation and testing of the protocol model should be independent of the simulation framework (in this case ns2). Only the installation and evaluation of protocols will necessitate use of the simulation framework.

Phased, Iterative Development

Once the developer understands the use cases for HCNeT, a development plan can emerge. A plan is necessary to meet the needs of the HCRG by a preset deadline. A

phased development plan is selected where completion of each phase represents a milestone for the customer. For HCNeT, a milestone represents the completion, at least in part, of one use case scenario. The product of each milestone will be well-documented, working software. Figure 10 illustrates the project development plan.

Within each phase, development will occur iteratively. In this way, a preliminary design is coded, the code is unit tested then refinements of the design are considered. The design/code/test iterations will occur until all requirements in the use case are met or until the end of the phase is reached. Each phase will last 2-3 weeks. If a milestone is not fully realized, the subsequent phases will be adjusted to incorporate the remaining requirements.

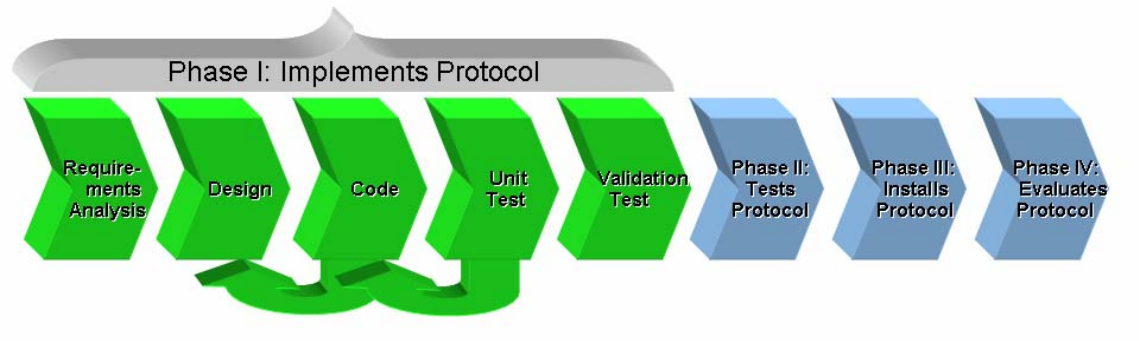

Figure 10. HCNeT Phased, Iterative Development Plan. Phase I is expanded out to demonstrate the activities for each phase. Design, Code and Unit Test activities occur in gradient parallel, where Design is the primary activity earlier and Unit Test is primary activity later. Each phase will last 2-3 weeks.

The advantage of the phased approach is concentration on producing a useful software package, even if it does not realize all the requirements. Additionally, by developing iteratively in each phase, the developer is not constrained to one design, but is able to evolve the design as fine-grained coding reveals challenges and opportunities.

The short development cycles are designed to avoid spending long periods of time coding an end-to-end design only to find the design must be reworked or discarded.

Phase I Design

To begin design on HCNeT, the high-value and high-risk requirements should be scrutinized first. Through use-case analysis, two requirements emerge that will help define the overall design of the system.

The designer is encouraged by the work of Battenfield [2] who developed a modular, hot-swappable routing framework for imbedded mobile systems. In [2] routing protocols were encapsulated into modules, then switched out as needed by the user or the application. HCNeT requirements add significant complexity to this concept since at least two protocols, routing and topology control, must switch out in almost any combination while still working in tandem. The design for HCNeT proceeds independently of Battenfield.

Firstly, of high risk and high value is providing an interface into Ns2 for the toolkit. As stated previously, Ns2 is a complex framework that is not easily abstracted. Conceptually, a protocol developer will build a network model with sufficient state for the operation of their protocol. Then, the developer will implement the protocol algorithm. When the algorithm is called, the network state will presumably change. This change must then be injected into the simulation automatically, without any explicit knowledge of Ns2. To encapsulate and hide the knowledge of Ns2, a general interface must exist to represent the operations expected for any simulation software. From that interface, a specific implementation will interact with Ns2. This line of reasoning points to the use of a bridge. The bridge public interface will emerge with the implementation

of the toolkit, then the Ns2 bridge class will adapt to Ns2 specifically. The bridge concept for HCNeT is illustrated in Figure 11.

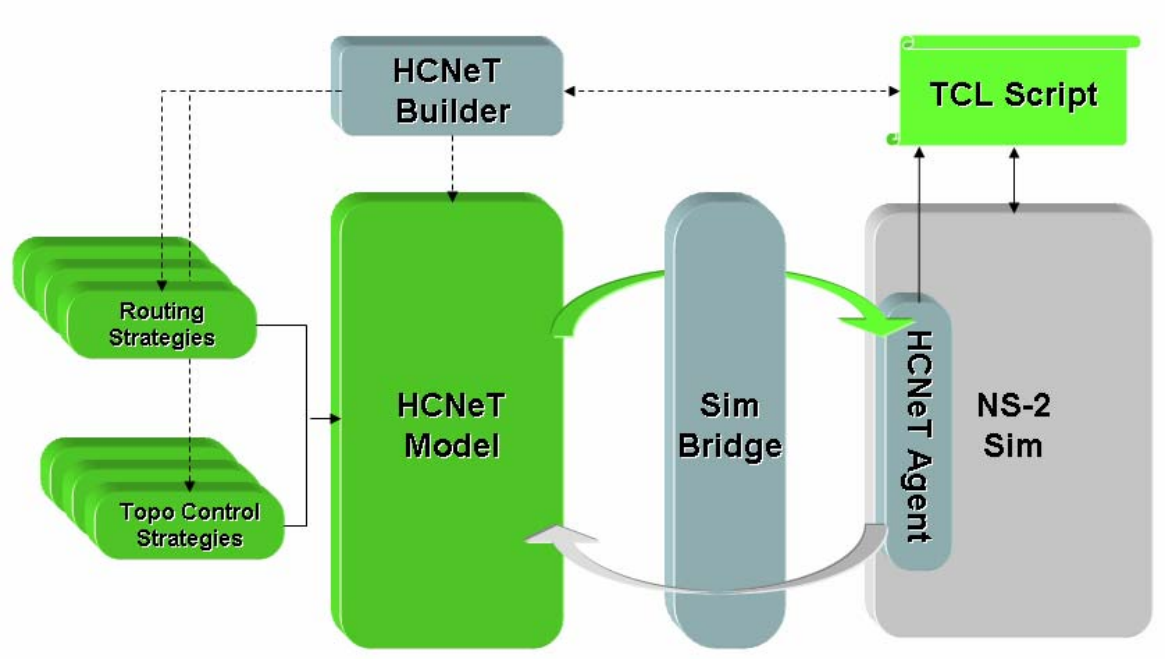

Figure 11. HCNeT Architectural Overview. Multiple user-defined protocol strategies operate on the user-defined network model. The user writes a TCL script that populates the model and controls the Ns2 simulation. The simulation passes HCNeT messages via the agent through the bridge. The model returns state to the agent through the bridge.

With a general architecture under consideration, the software developer must put thought into how the model to simulation communication process will take place. If a desired process can be modeled using the selected architecture, then the architecture should be considered a feasible approach by the developer. HCNeT general communication flow is modeled by the sequence diagram in Figure 12 and indicates a feasible architecture.

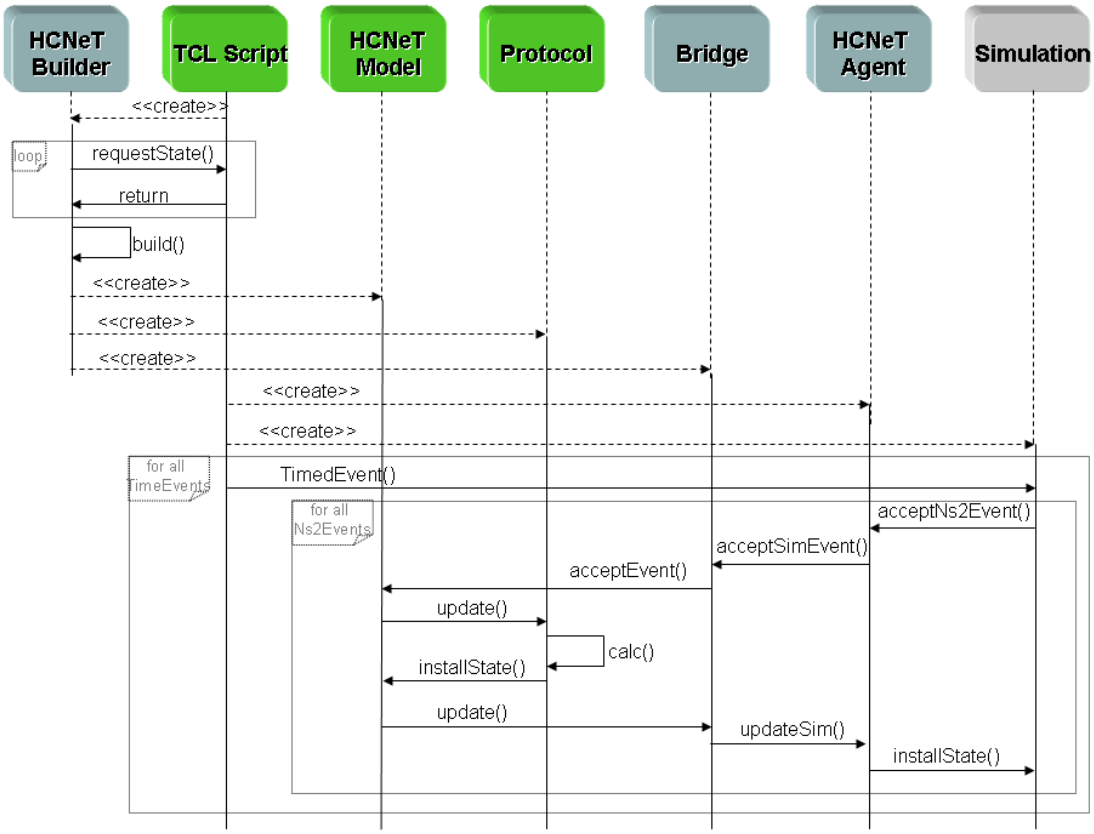

Figure 12. HCNeT Sequence Overview. A conceptual ordering of events in an HCNeT run-time scenario. Through process modeling, an order of creation emerges. The general order of message calls for event handling is also established.

With one risky design consideration thought through, the toolkit designer must address another potentially risky and high value feature. From the requirements analysis, the designer knows the toolkit must be flexible enough to be extended in different ways to support specific protocols. The decorator software design pattern seems to be able to support dynamically selectable views of an object. Since the decorator realizes a composite structure, each protocol can traverse the model structure and select which view it requires through the visitor pattern.

Finally, the HCNeT designer considers the need for hot-swappable protocol implementations within a family of protocols. For example, PPRP routing could be

switched in favor of CBRP. This requirement exemplifies the intended use of the strategy design pattern.

Phase I Implementation

As previously stated, the implementation phase will most likely face the most significant challenge interfacing the network toolkit with Ns2. Like [15], the ns2 Agent class will serve as the access point to the ns2 framework. The agent should be reached through a bridge class. The agent is a shell Ns2 protocol representing HCNeT as in Figure 8 and Appendix B that will allow interception and manipulation of simulation events during run-time. As per [5], notice the need to create special TCL binding classes in Ns2 to ensure correct interaction within the split framework. Additionally, the coder modified four other Ns2 configuration files to successfully introduce the protocol.

The toolkit designer began implementation of the HCNeT model design decisions as seen in Figure 13. NetworkComponent, BasicNetwork and NetworkExtension comprise the decorator pattern. New decorations, here called network extensions, implement NetworkExtension. With an eye toward integrating Llewellyn's protocol, the coder implemented the ClusteredExtension class that will hold all the state and methods required by a topographically clustered network. Notice how clustered network state such as cluster heads and gateway nodes may be implemented using the basic elements of the toolkit, such as the HNode class. Because the design calls for a visitor interface, the coder created the NetworkComponentVisitor class and added the accept() method to the NetworkComponent class. Source code for Hcnet is included in Appendix B.

The implementation will evolve as each design decision is considered for coding and the legacy protocol code is integrated. Work must be completed to implement the

bridge class and layer in references to the simulation bridge within the NetworkComponent public interface. Ideally, the observer pattern will be used to decouple the Ns2 based bridge from the toolkit.

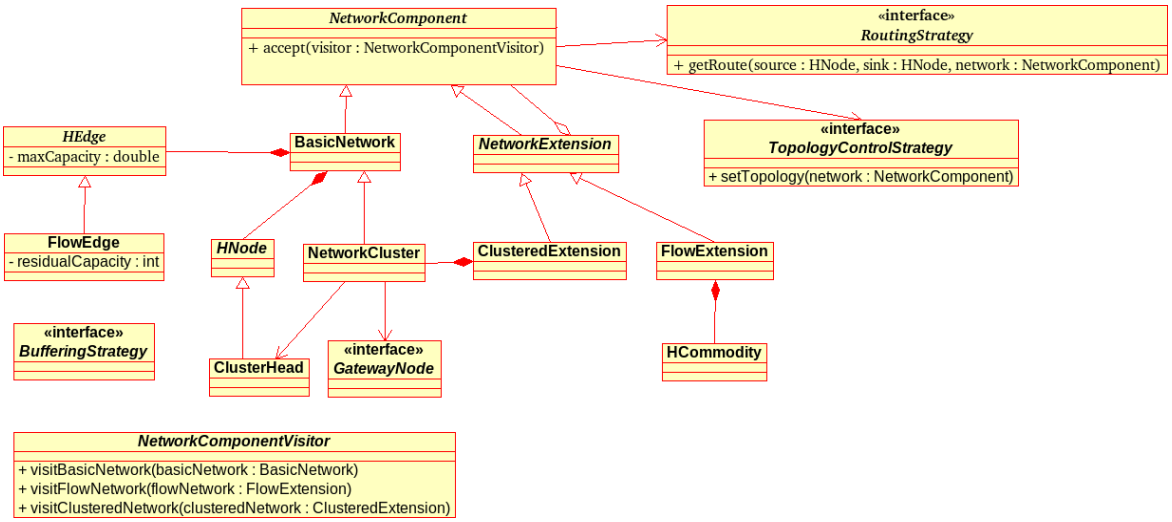

Figure 13. Class model for HCNeT network toolkit. This model is extensible and configurable, relying mainly on the Decorator pattern and the Strategy pattern.

IV. Results

Architecture

Network Model

The network model contains a strategy interface for routing and a strategy interface for topology control as planned. Metadata and method calls required for new protocols will be placed in user-defined strategy implementations. The key feature of the strategies is they may be switched out independently and dynamically to accommodate research requirements.

The network topology model was implemented per the original design decision and is depicted in **Figure 14**. The decorator pattern implementation allows for user-defined network structures to be dynamically applied and removed as required by any particular routing or topology control protocol. This implementation ensures correct interoperability of various protocols throughout the execution time of the simulation.

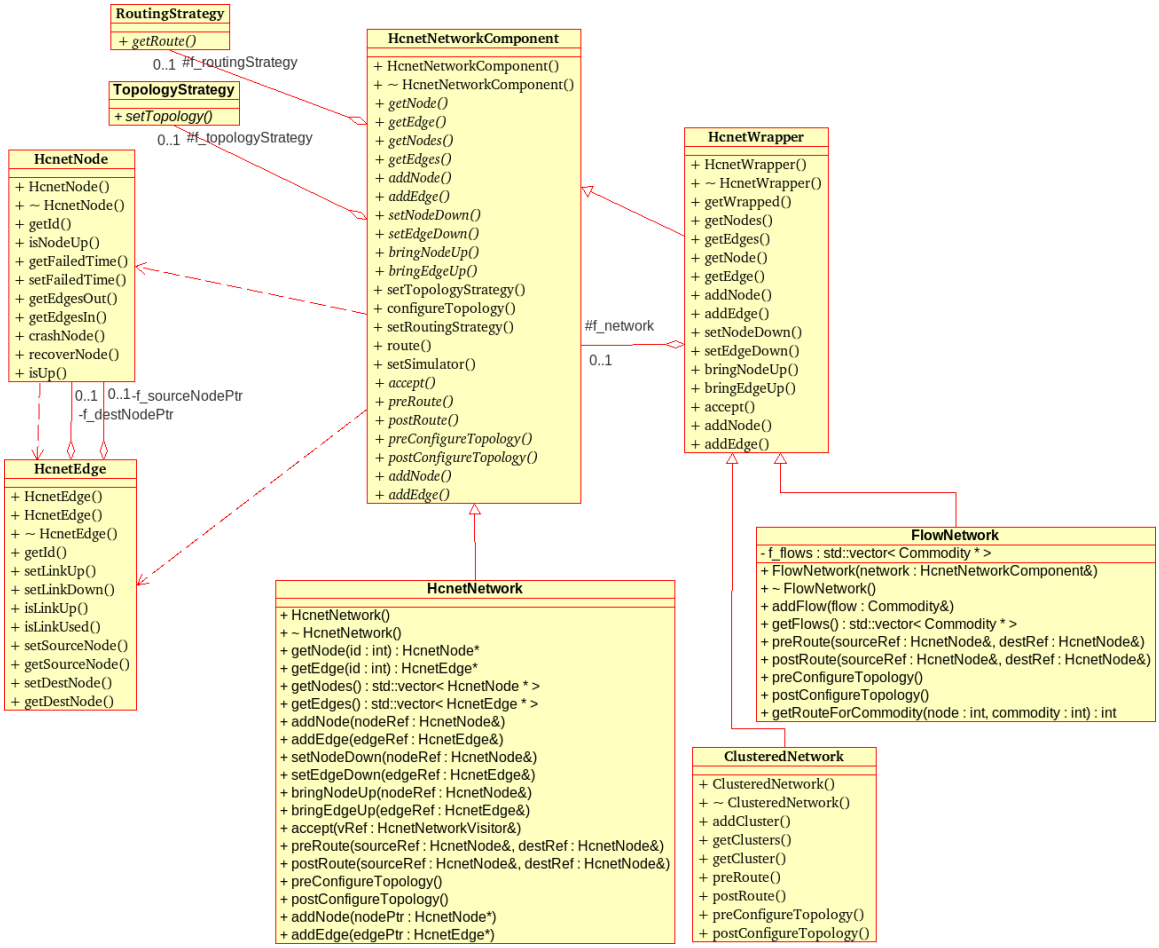

Figure 14. HCNeT Toolkit Network Model. The decorator pattern (wrapper) is applied to the network container class, meaning HcnetNetwork may be wrapped by FlowNetwork and/or ClusteredNetwork at any time during execution.

Builder

The builder consolidates the construction of the network model. For implementation of Llewellyn's protocol, the builder depicted in **Figure 15** currently reads a formatted TCL script much like an XML file. This was done for convenience since much of the work to populate the network model was already implemented in various portions of the legacy code.

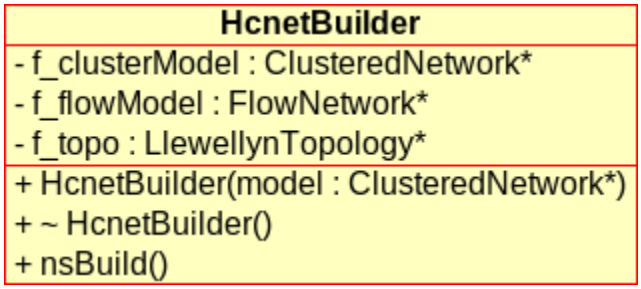

Figure 15. HcnetBuilder Class.

Bridge

The bridge depicted in **Figure 16** evolved to be a two-way bridge since the toolkit side contains a reference to the toolkit and the Ns2 side contains a reference to the agent hq. Therefore, the bridge is actually more of a logical mediator.

The bridge always has the same interface to the toolkit model and is to be treated as a proxy for the simulation engine when viewed by the toolkit model. As a singleton, the Bridge is observed globally and allows for protocol updates to the simulator on demand. For ns2 the updates are performed through the Tcl instance by sending commands.

On the simulator side, the bridge contains a set of observers that are updated with each clock cycle, so each clock cycle the toolkit has an opportunity to change the state of the network. Most notably in this implementation, the network nodes are registered observers and thus are updated each simulation second.

The Bridge may be implemented for any simulation engine such as ns2 or OPNeT. The simulator bridge was implemented as an abstract singleton as designed, therefore the bridge contains a registry of specific implementations. The specific implementation is selected through a system environment variable at load time.

Ns2

For the ns2 implementation, the EPOCHS AgentHQ was used as a starting point. Ultimately the AgentHQAgent class was refactored to communicate with the bridge. Additionally, AgentHQAgent served as a proxy to the network model for the SrcDestFidHashClassifier Class. SrcDestFidHashClassifier serves as a representative for each toolkit node within Ns2. Only these two Ns2 classes have any knowledge of the toolkit within the simulator. Through access to a TCL instance, the AgentHQAgent and SrcDestFidHashClassifier are able to transmit commands required by the toolkit to the Ns2 environment.

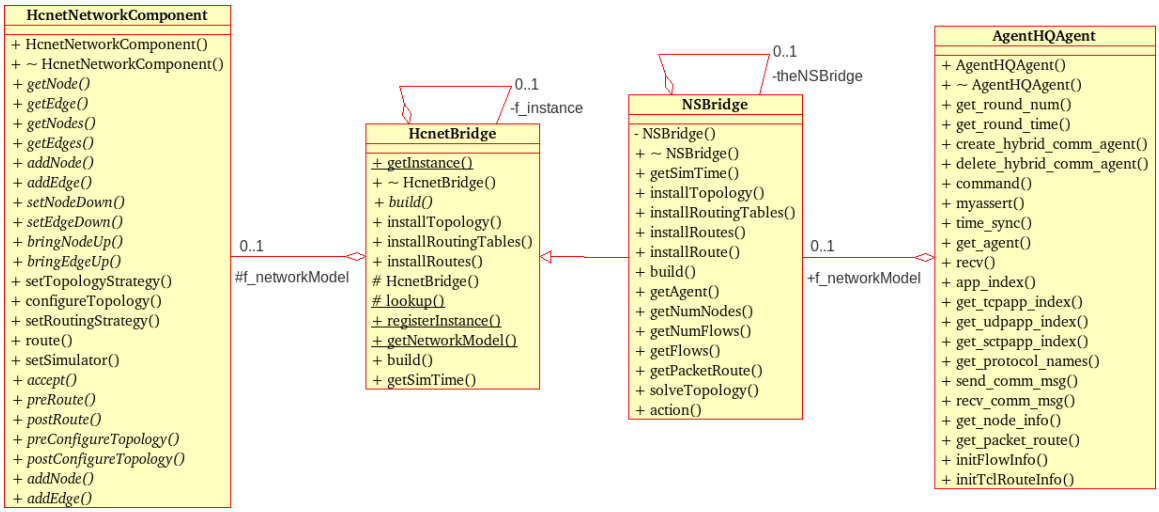

Figure 16. HCNeT Bridge Interfaces HCNeT Network Model with Ns2 Agent. Use of Abstract Singleton pattern ensures network model is unaware of which simulation application is being used.

Routing and Topology Control Implementation

To demonstrate utility of HCNeT, the developer embarked on a project to refactor the routing topology control protocol implemented by Llewellyn [15]. Convincing progress illustrates that only four toolkit classes need to be extended to exhibit the

protocol. The node, edge, and commodity classes required extension to hold the state required by Llewellyn's algorithm. Of course, the TopologyStrategy class required extension to contain the logic of the protocol. **Figure 17** depicts the four new classes.

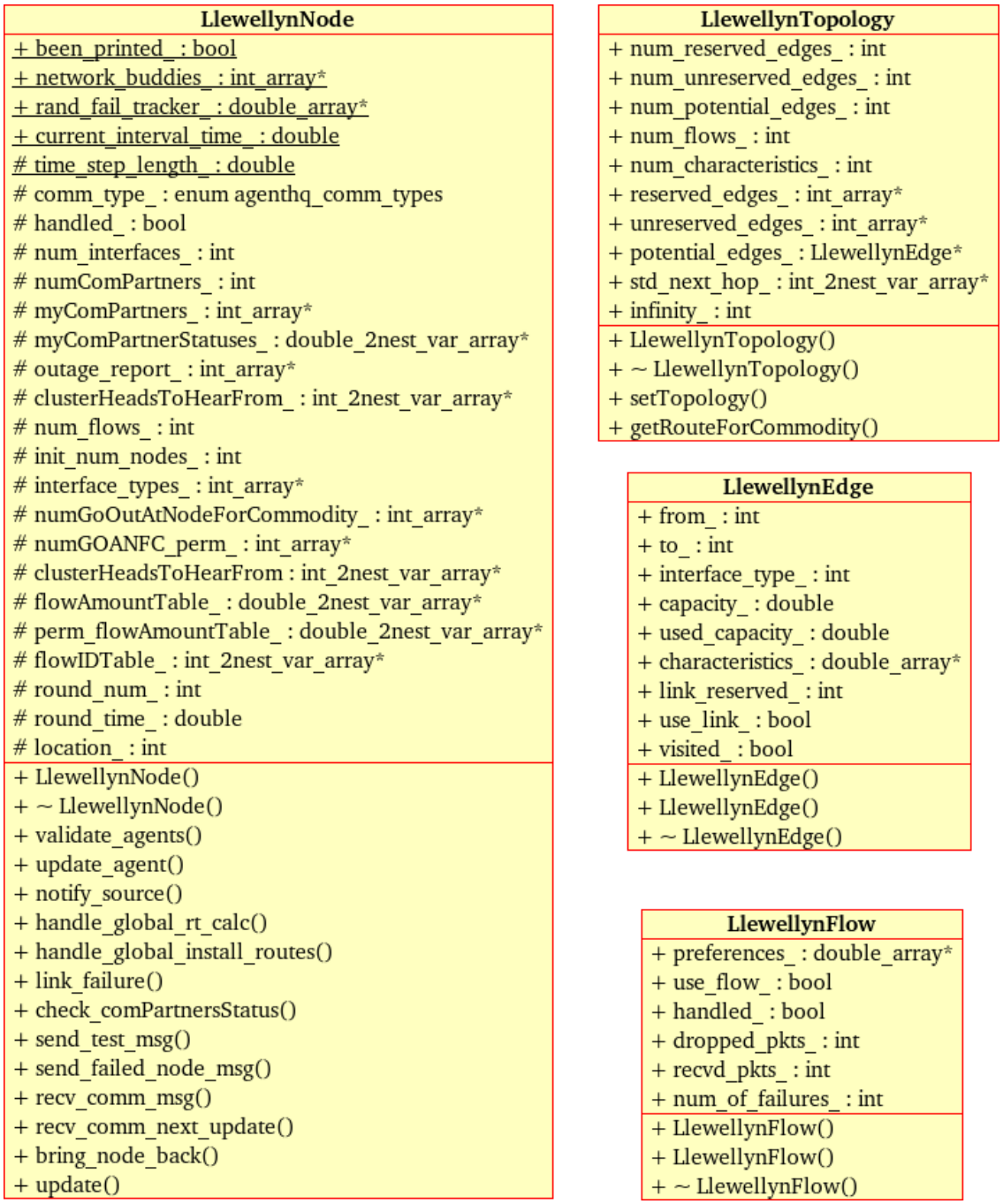

Figure 17. Class Definitions for New Protocol Developed in HCNeT. Llewellyn's routing and topology control algorithm is implemented in HCNeT by extending four toolkit classes.

Maintainability

One of the important goals of software development is to produce software that is not resistant to change.[3] Object Oriented (OO) design principles enforce the encapsulation and cohesiveness of software modules. By applying these principles in a disciplined manner, such as emulating software design patterns, understandability and changeability of the software increases. HCNeT was designed using well known OO design patterns[8] and implemented using object features of the C++ programming language. Code analysis reveals key OO indicators and complexity factors support the claim of HCNeT maintainability.

A description of some measures of maintainability follows and Table 1 summarizes these measures as indicators of maintenance qualities. Lastly, an analysis of HCNeT as compared to a previous network protocol project in ns2 using these measurements.

Coupling

Coupling describes the direct dependency between software objects. When one object contains a reference to another object, this is an instance of coupling. High coupling refers to objects that exhibit this containment property in excess. Excessive coupling can be an indication of poor encapsulation; the result of which are objects that depend highly on other objects and thus, increasing the effect of change on one object to others. High coupling leads to poor maintainability.

Number of Children

Number of children refers to size of the first level of the inheritance tree of some super class. A large number of children for a given super class can indicate the loss of specialization or inappropriate abstraction in the design[14]. However, in some cases, such as application of the observer pattern[8], a large number of children may not indicate a maintainability problem.

Depth of Inheritance Tree

The depth of a class' inheritance tree is a consideration for maintainability. On one hand, a particularly long path of inheritance could lead to understandability difficulty as a software specialist attempts to comprehend the gap between the top and the bottom of the tree. On the other hand, a long path of inheritance exhibits great potential for reuse of ancestor classes. The key for this metric is to analyse the intent of large inheritance trees to determine if the appropriate balance of understandability and reuse has been made.[14]

Composite Information Flow

Composite information flow is a function of the fan-in and fan-out of a software module. Fan-in refers to the number of other modules which change the state or invoke operations on a particular module. Fan-out refers to the number of modules that read the reference modules state combined with the number of modules the reference module operates on. [14]

A module with a high composite information flow implies the need to recompile client modules if supplier modules are changed.[14]

Cyclomatic Complexity

McCabe's cyclomatic number can be viewed as a direct measure of the number of independent paths through the procedural logic of a subprogram, but it is more commonly viewed as an indirect measure of maintainability, and particularly of testability. The McCabe cyclomatic number also has a particularly direct relationship to the testability of a component. [14]

Table 1. Relationships between Metrics and Quality Model Attributes.[14]

	self-containedness	self-descriptiveness	structuredness	conciseness	legibility	augmentability	abstractness	reusability	buildability	Notes
CBO	X	X		X						Coupling Between Objects
NOC						X	X	X		Number of Children
DIT					X			X		Depth of Inheritance Tree
IF4	X					X	X	X	X	Composite Information Flow
MVG			X	X	X					McCabe's Cyclomatic Complexity

Maintainability of HCNeT

When Littlefair's metrics and 3-tier evaluation [14] are applied to HCNeT as seen in Table 2, all but the coupling metric reveal a maintainable profile. The coupling metric is found suspicious in two classes, Network and Node. However, on inspection, in the network object model, the network container and the node are central elements and

coupling was designed for ease of access to these core classes. Even though there are only two classes with suspicious coupling, one may consider refactoring in this case.

AgentHQ, the ns2 interface structure utilized in [10] and [15] are provided in Table 2 for comparison.

Table 2. Maintainability Measurements, HCNeT vs AgentHQ

Key Maintainable Suspicious Design Flaw	AgentHQ	HCNeT
Coupling Between Objects (CBO)	All < 11	Network 16, Node 17, Others < 7
Number of Children (NOC)	All < 2	All < 2
Depth of Inheritance Tree (DIT)	All < 4	All < 3
Composite Information Flow (IF4)	All < 64	All < 9
McCabe's Cyclomatic Complexity (MVG)	637	75

V. Conclusion

HCNeT provides an extensible toolkit for quickly implementing and testing a combination of network protocols at a more theoretical level. Once implemented, the toolkit will be able to demonstrate the combined performance of two topology control and routing algorithms in a clustered network. Immediately thereafter, the HCRG can utilize the toolkit to rapidly prototype other network protocols.

Contributions

1. Requirements analysis led to an overall architectural design approach for a general purpose network simulation toolkit for use in network protocol development research.
2. Laid the groundwork for a simulation toolkit providing course-grained rapid prototyping along side with fine grained protocol implementation.
3. Implemented a bridging construct to abstract away specific network simulation application to allow for “plug and play” testing of new protocols.
4. Agile implementation process fleshed out a robust OO design and provided a maintainable code base available for future toolkit developers. OO-driven self-documenting code provides intuitive code refactoring.

Future Work

A completed refactoring of Llewellyn’s and Garner’s protocol is necessary to demonstrate and analyze the performance of the various algorithms working in concert.

New work must be completed to build bridge plugins that will allow HCNeT to be adapted to a variety of network simulation frameworks, in particular OPNET.

The toolkit developer will proceed with Phases II-IV of the project, making HCNeT more user-friendly. A unit testing suite must be developed to provide assurance to protocol developers of correct toolkit implementation. Ultimately, a GUI like the one for SynTraff [1] should be provided to allow for easy install, build and execution of user-selected routing and topology control protocols, and simulation engines.

Appendix A: HCNeT Use Cases

Global Requirements:

System

HCNeT requires the use of ns2. See ns2 for more details on platform requirements and installation.

Experience/Knowledge

User:

A user of HCNeT is familiar with the ns2 framework in general and, in particular, TCL programming language which is used as a front-end to build network architectures and simulation parameters.

Developer:

A developer of HCNeT has the same characteristics of a user and additionally is experienced in OO programming with C++.

Use Case: H_UC01 Implement a New Network Protocol

CHARACTERISTIC INFORMATION

Goal in Context: A protocol developer codes the protocol using HCNeT

Scope: C++ development environment

Level: Primary

Preconditions: C++ compiler is installed, HCNeT header and source files are installed within the programming scope.

Success End Condition: Protocol developer instantiated classes or user-extended classesfrom within HCNeT to fully express the protocol logic.

Failed End Condition: Protocol developer could not fully express the protocol logic by instantiating or simply extending HCNeT

Primary Actor: Protocol developer

Trigger: Protocol developer completes conceptual design and is ready to implement

---

MAIN SUCCESS SCENARIO

1. User selects an HCNeT generic protocol class or interface and programs his/her particular protocol logic in compliance with the generic interface.
2. User successfully compiles their new protocol source code

EXTENSIONS

1a. An appropriate generic protocol does not exist in HCNeT

1a.1. User creates an appropriate generic protocol

1a.2. User creates an appropriate extension to the network model to support the generic protocol

2a. Source code will not compile

2a.1. Go to use case Test HCNeT

SUB-VARIATIONS

1. Default generic protocols are Routing, TopologyControl and Buffering

---

RELATED INFORMATION

Priority: 1

Performance Target: <the amount of time this use case should take>

Frequency: Frequent

Superordinate Use Case: <optional, name of use case that includes this one>

Subordinate Use Cases: Test HCNeT

OPEN ISSUES

SCHEDULE

Due Date: TBD

Use Case: H_UC02 Test HCNeT

CHARACTERISTIC INFORMATION

Goal in Context: A protocol developer tests their implementation

Scope: unit testing

Level: Subfunction

Preconditions: Use Case Implement a Network Protocol

Success End Condition: Protocol developer knows their protocol implementation is compatible with the HCNeT model or knows why it is not compatible

Failed End Condition: Protocol developer does not know if their protocol is not compatible with the HCNeT model

Primary Actor: Protocol Developer

Trigger: Protocol developer wishes to unit test HCNeT

MAIN SUCCESS SCENARIO

1. User selects to unit test HCNeT

2. HCNeT informs user that all unit tests passed succesfully

EXTENSIONS

2a. HCNeT informs user that one or more of the unit tests have failed and why

    2a.1. Go to use case Implement a New Network Protocol.

SUB-VARIATIONS

(No known sub-variations)

RELATED INFORMATION (optional)

Priority: 2

Performance Target:

Frequency: Frequent

Superordinate Use Case: Implement a New Network Protocol

Subordinate Use Cases: none

OPEN ISSUES

SCHEDULE

Due Date: TBD

Use Case: H_UC03 Install HCNeT

CHARACTERISTIC INFORMATION

Goal in Context: User installs HCNeT and/or new user protocols into the ns2 framework

Scope: Compiler

Level: Primary Task

Preconditions: Test HCNeT main success scenario

Success End Condition: HCNeT and ns2 compile together

Failed End Condition: compiler errors

Primary Actor: Protocol developer

Trigger: Protocol developer ready to run simulation

MAIN SUCCESS SCENARIO

1. User runs “make all” in the ns2 top directory

2. Compiler informs user of successful install

EXTENSIONS

2a. Compiler informs user of error(s)

    2a1. User writes new unit test

    2a2. Use Case Test HCNeT

    2a3. If all else fails, get help from AFIT staff

SUB-VARIATIONS

(no known sub-variations)

RELATED INFORMATION

Priority: 1

Frequency: Frequent

Superordinate Use Case: none

Subordinate Use Cases: none

OPEN ISSUES

SCHEDULE

Due Date: TBD

Use Case: H_UC04 Evaluate Protocol(s)

CHARACTERISTIC INFORMATION

Goal in Context: In the course of defining a network simulation, a user declares a topology control protocol to override the default or previously declared protocol.

Scope: TCL scripting

Level: Primary

Preconditions: HCNeT is installed, a text editor is open and a shell tcl script is written.

Success End Condition: ns2 will simulate network operation based on the user defined protocol. Successful completion will result in a success message in the simulation trace file and a graphical indication in the nam viewer.

Failed End Condition: The previously declared protocol will continue. Failed completion will result in a failure message in the simulation trace file and a graphical indication in the nam viewer.

Primary Actor: user

Trigger: user decides when to declare the protocol

MAIN SUCCESS SCENARIO

Step 1: User types the HCNeT protocol declaration into the tcl script

Step 2: User saves the script and executes the script with ns2

Step 3: User executes the nam file with nam

Step 4: User observes effects of the protocol, with a visual indication of when the protocol is active.

EXTENSIONS

Step 4: Protocol fails: User observe exception message and begins troubleshooting

---

SUB-VARIATIONS

Step 1: user may select a specific topology control or routing protocol

Step 3a: nam is not used, user examines trace file

Step 3b: this step may be programmed into the TCL script

RELATED INFORMATION (optional)

Priority: 1

Performance Target: <the amount of time this use case should take>

Frequency: Most Frequent

Channel to primary actor: installed HCNeT used through text editor of choice

---

OPEN ISSUES

Is this use case sufficient for the vision of HCNeT?

SCHEDULE

Due Date: TBD

Bibliography

[1] Balakrishnan, R., and C. Williamson. "The synTraff Suite of Traffic Modeling Toolkits." *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems – Proceedings*. 2000.

[2] Battenfield, O. and others. "A Modular Architecture for Hot Swappable Mobile Ad hoc Routing Algorithms." *IEEE Proceedings of the Second International Conference on Embedded Software and Systems*. IEEE, 2005.

[3] Booch, G. and others. *The Unified Modeling Language User Guide, 2d Ed*. Addison-Wesley, Upper Saddle River, NJ, 2005.

[4] Cayne, B. and D. Lechner. *New Webster's Dictionary and Thesaurus of the English Language*. Lexicon Publications, Inc, Danbury, CT, 1993.

[5] Chung, J and M. Claypool. "NS by Example." Worcester Polytechnical Institute, MA, 2002.

[6] Fall, K. "The ns Manual." The VINT Project. Berkely, CA, 2007.

[7] Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, MA, 2003.

[8] Gamma, E. and others. *Design Patterns*. Addison-Wesley, Boston, MA, 1995.

[9] Garner, L. *Heuristically Driven Search Methods for Topology Control in Directional Wireless Networks*. MS Thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2007.

[10] Hopkinson, K. and others. "EPOCHS: A Platform for Agent-Based Electric Power and Communication Simulation Built From Commercial Off-the-Shelf Components." *IEE Transactions on Power Systems*, Vol 21, No 2 (May 2006).

[11] Karl, M. "A Comparison of the Architecture of Network Simulators Ns2 and TOSSIM." Student Project. Institute for Parallel and Distributed Systems, University of Stuttgart. Jan 2005.

[12] Larman, C. *Applying UML and Patterns*. Prentice Hall PTR, Upper Saddle River, NJ, 2006.

[13] Lin, S., et al. "WiDS: an Integrated Toolkit for Distributed System Development." Downloaded at USENIX.com (April 2007).

[14] Littlefair, T. *An Investigation into the Use of Software Code Metrics in the Industrial Software Development Environment*. PhD thesis. Faculty of Communications, Health and Science, Edith Cowan University, 2001.

[15] Llewellyn, L. *Distributed Fault-Tolerant Quality of Service Routing in Hybrid Directional Wireless Networks*. MS Thesis. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2007.

[16] Metzger, S. and W. Wake. *Design Patterns in Java*. Addison-Wesley, Boston, MA, 2006.

[17] State, R. and others. "An Extensible Agent Toolkit for Device Management." *IEE Web Services and Management*, IEEE (2004).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
27-03-2008	Master's Thesis	Apr 2007 – Mar 2008

4. TITLE AND SUBTITLE

A Hybrid Communications Network Simulation-Independent Toolkit

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dines, David, M., Major, USAF

5d. PROJECT NUMBER
ENG 08-175

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/08-08

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Office of Scientific Research
David Luginbuhl, PhD
875 N Randolph Street
Ste 325 Rm 3112
Arlington, VA 22203-1768
(703) 696-6565 (DSN: 426-6207), e-mail: david.luginbuhl@afosr.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR/NM

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Net-centric warfare requires information superiority to enable decision superiority, culminating in insurmountable combat power against our enemies on the battlefield. Information superiority must be attained and retained for success in today's joint/coalition battlespace. To accomplish this goal, our combat networks must reliably, expediently and completely deliver over a wide range of mobile and fixed assets. Furthermore, each asset must be given special consideration for the sensitivity, priority and volume of information required by the mission. Evolving a grand design of the enabling network will require a flexible evaluation platform to try and select the right combination of network strategies and protocols in the realms of topology control and routing. This research will result in a toolkit for ns2 that will enable rapid interfacing and evaluation of new networking algorithms and/or protocols. The toolkit will be the springboard for development of an optimal, multi-dimensional and flexible network for linking combat entities in the battlespace

15. SUBJECT TERMS
network, simulation, toolkit, ns2, software, design

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Kenneth Hopkinson, PhD (ENG)
U	**U**	**U**	**UU**	65	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4579 (khopkinson@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18